Graphics Programming

CSE/IT 213

NMT Department of Computer Science and Engineering

"Nobody actually creates perfect code the first time around, except me. But there's only one of me."

— Linus Torvalds

"We had to attend a yearly lecture on the Russian language. Each of the students made it a point of honour to have the worst results in that subject. I did not answer a single question."

— Waclaw Sierpinski

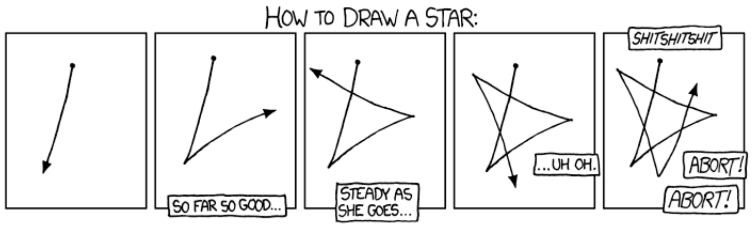


Figure 1: `https://xkcd.com/1029/`

Problems

Problem 1: Draw Shape

Create a small helper class, `DrawShape.java`, to draw shapes onto a `Graphics2D` surface. You will use this code in the next problems. You will have to import classes from your geometry package, last seen in Homework 5. The class contains three static methods:

DrawShape
+ drawPoint(Graphics2D, Point, Color) : void + drawRectangle(Graphics2D, Rectangle, Color) : void + drawCircle(Graphics2D, Circle, Color) : void

`drawPoint()` takes a `Point` and draws a point at that position on the `Graphics2D` object. Draw the point using the given color, but set the color back to its original value afterward (save a copy of the original color before you start drawing).

Note: The builtin `Graphics2D` object doesn't have a `drawPoint()` method. Instead, you might use `drawLine()` to draw a line of length 1, or `drawOval()` to draw a circle with a radius of 1 – the implementation is up to you.

`drawRectangle()` takes a `Rectangle` and uses the `Graphics2D`'s `drawRect()` method to draw its outline to the surface. Draw the rectangle using the given order, but set the color back to its original value after drawing it.

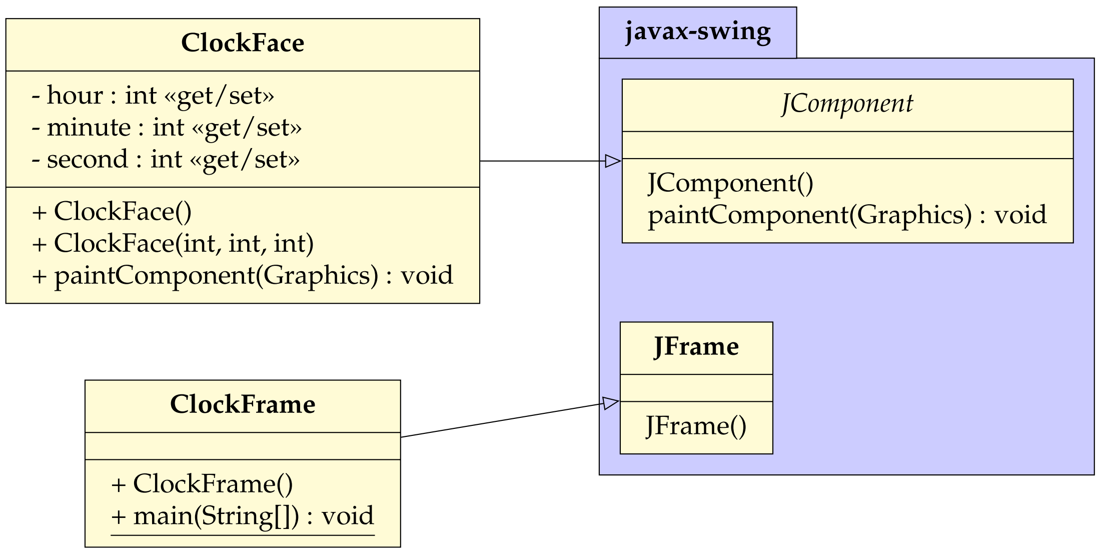
Note: The arguments to `drawRect()` are `(x, y, width, height)`, not `(x1, y1, x2, y2)`.

`drawCircle()` takes a `Circle` and uses the `Graphics2D`'s `fillOval()` method to draw the circle in the given color. Draw the rectangle using the given order, but set the color back to its original value after drawing it.

Note: The `fillOval()` method takes arguments `(x, y, width, height)`, but `x` and `y` represent the lower left corner of the oval's bounding box, *not* the center!

Problem 2: Clock

For this problem, you will write a program to display the current time with an analog clock. You will need to download the image `clock.jpg` from Canvas and add it to your project.



Clock Face

First create the class `ClockFace.java`, which inherits from `JComponent`.

The class has three private members, `hour`, `minute`, and `second`. The default constructor should set their values to match the current local time, which you can get using the method `LocalTime.now()`. Also set the preferred size of the `JComponent` to 400×400:

```
setPreferredSize(new Dimension(400, 400));
```

The second constructor should do the same, but set the attributes according to its three input arguments. Check that `hour` is between 0 and 23, and that `minute` and `second` are between 0 and 59. If any value is out of range, then set that value with the current local time instead.

The `paintComponent()` method modifies the `JComponent` to display a clock face. The first step is to read in the background image:

```
BufferedImage bg = ImageIO.read(new File("clock.jpg"));
```

Then draw it to the frame with `drawImage(bg, 0, 0, null)`. Next you will draw three line segments, one for each hand on the clock. In order to differentiate between them, each hand should be drawn with a different length:

$$r_{sec} = 190 \qquad r_{min} = 160 \qquad r_{hr} = 100$$

Cast the given `Graphics` argument to a `Graphics2D` object. And use it set different stroke widths for each line segment, and set the color of the second hand to red.

These equations will help you determine the *angle* each hand will make with the 12 o'clock position on the clock:

$$\theta_{sec} = \frac{\pi}{30} \times S \qquad \theta_{min} = \frac{\pi}{30} \times M \qquad \theta_{hr} = \frac{\pi}{6} \times (H \bmod 12) + \frac{\pi}{360} \times M$$

You can then use these conversions to get the end-points of each line segment:

$$(x, y) = (200 + r \cdot \sin\theta, 200 - r \cdot \cos\theta)$$

Clock Frame

Now create the class `ClockFrame.java`, which inherits from `JFrame`. The constructor should set up the main window, and add a `ClockFace` to the set of window components.

Also add a `JLabel` above the clock image that displays the time zone of the clock. You can get the current time zone with `TimeZone.getDefault()`, and use its `getDisplayName()` method to get a human readable name. You can add an option to the `JLabel` constructor to center the text:

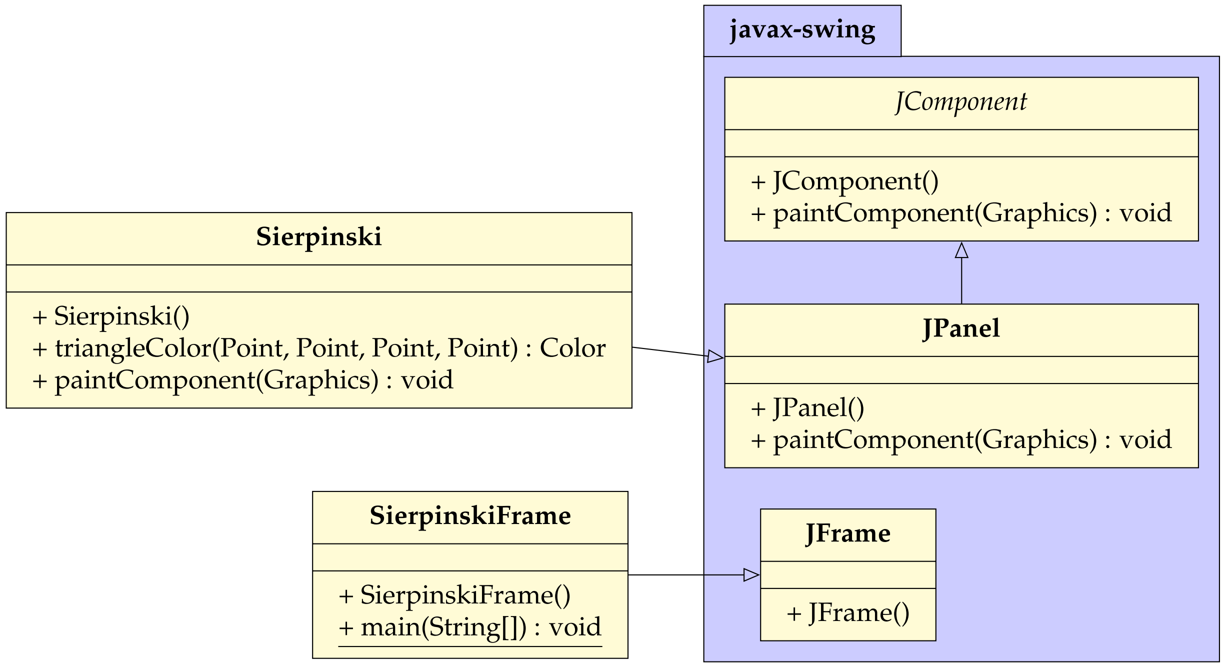
```
new JLabel(tzName, SwingConstants.CENTER);
```

The font should be set to sans-serif, 24pt, bold. Add the label to the top of the `JFrame`, immediately `BorderLayout.NORTH` of the clock.

Write a `main()` method that creates a new `ClockFrame`, and opens the window to display the time. When the user closes the window, the program should exit.

Problem 3: Sierpinski

In this problem you will write a program to draw a fractal pattern called *Sierpinski's triangle*. You will have to import your `Point` class, last seen in homework 5.



javax-swing
JComponent
+ JComponent()
+ paintComponent(Graphics) : void
JPanel
+ JPanel()
+ paintComponent(Graphics) : void
JFrame
+ JFrame()
Sierpinski
+ Sierpinski()
+ triangleColor(Point, Point, Point, Point) : Color
+ paintComponent(Graphics) : void
SierpinskiFrame
+ SierpinskiFrame()
+ main(String[]) : void

Sierpinski

`Sierpinski.java` inherits from `JPanel`. `JPanel` is itself a subclass of `JComponent`, designed to act as a container for other components. For this program, the only important feature is that it allows you to set the background color:

```
setBackground(Color.BLACK);
```

The constructor should set the preferred size to 500×500, and set the background color to black.

`triangleColor()` converts a point to an RGB color. The idea is to represent red, green, and blue using three distinct coordinates. You choose a color for any point inside that triangle by measuring its distance to all three of these points. If L is the maximum distance between the points R, G, and B, (the longest side of the RGB triangle), then the point x will have the color:

$$r = \left\lfloor 255 \times \frac{L - \delta(x, R)}{L} \right\rfloor \qquad g = \left\lfloor 255 \times \frac{L - \delta(x, G)}{L} \right\rfloor \qquad b = \left\lfloor 255 \times \frac{L - \delta(x, B)}{L} \right\rfloor$$

`paintComponent()` uses the previous two methods to draw a the Sierpinski triangle. First create an array containing these three points: $R = (250, 77), G = (50, 423), B = (450, 423)$. Then you can then draw the fractal using this randomized algorithm:

1. Set the point x to one of R, G, and B
2. For some high number of iterations:
   (a) Randomly choose one of R, G, and B as a target point
   (b) Calculate the midpoint between x and the target
   (c) Set x to the value of the midpoint
   (d) Draw x

This algorithm (surprisingly) only selects points inside the fractal, so after many iterations the pattern emerges:

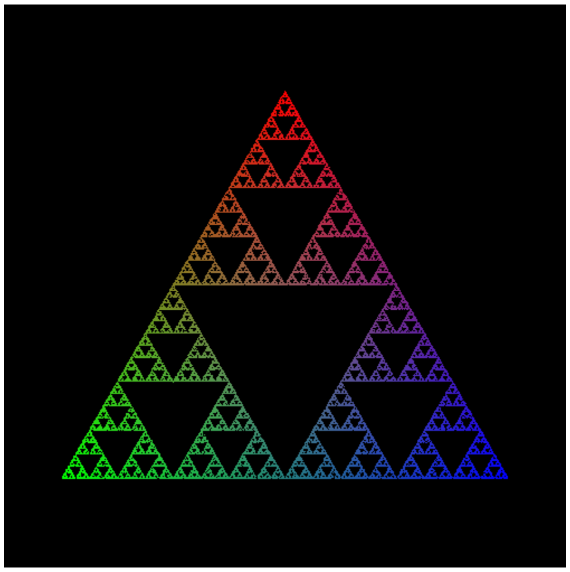


Figure 2: Result of the algorithm after 50,000 iterations

Use your `DrawShape.drawPoint()` to draw the points, and `triangleColor()` to decide their colors. You can use `new Random().nextInt(3)` to randomly select points from the array.

Sierpinski Frame

Finally, write `SierpinskiFrame.java`, which inherits from `JFrame`. The constructor should set up the main window, and add a `Sierpinski` panel to the set of window components.

Write a `main()` method which creates and displays a new `SierpinskiFrame`. The program should exit when the window is closed.

Problem 4: Go Board

In this problem you will use Java's graphics tools to visualize data. You will have to import your `Rectangle` and `Circle` classes from homework 5. You also need to download `bamboo.jpg` and `stones.txt` from Canvas and add them to your project.

GoBoard
- stones : Color[][]
+ GoBoard() + fromFile(String) : GoBoard + addStone(int, int, Color) : void + paintComponent(Graphics) : void

GoFrame
+ GoFrame(GoBoard) + main(String[]) : void

Go Board

The important class, GoBoard.java, inherits from JComponent. The constructor should set the preferred size to 720×720. It also initializes the private attribute stones to be an empty 19×19 array of Colors, (each entry is initially set to null).

addStone() takes a pair of indices, i and j, and uses them to set stones[i][j] to the given color. If either index is outside the range from 0 to 18, do nothing.

The static method fromFile() takes a file path, and reads the given file to add stones to a new GoBoard [1]. The file can be expected to have the following format:

```
...
B D 9
B M 17
W E 7
B S 3
B G 12
...
```

The first column indicates the stone's color, either Color.BLACK or Color.WHITE. The second column is a single capital letter between 'A' and 'T', not including 'I'. The last column is a number from 1 to 19. The last two fields indicate the column and row of the board, respectively.

Read the file, and convert each letter and number to an index in the array (subtract 1 from the column if col.charAt(0) > 'I'). Add a stone of the given color to that index in the new GoBoard's array. When you get to the end of the file, close it and return the GoBoard. If any IO or Scanner operation throws an exception, catch it and throw an IllegalArgumentException instead.

Finally, write a paintComponent() method to show the contents of the array. First, open bamboo.jpg and set it as the background image.

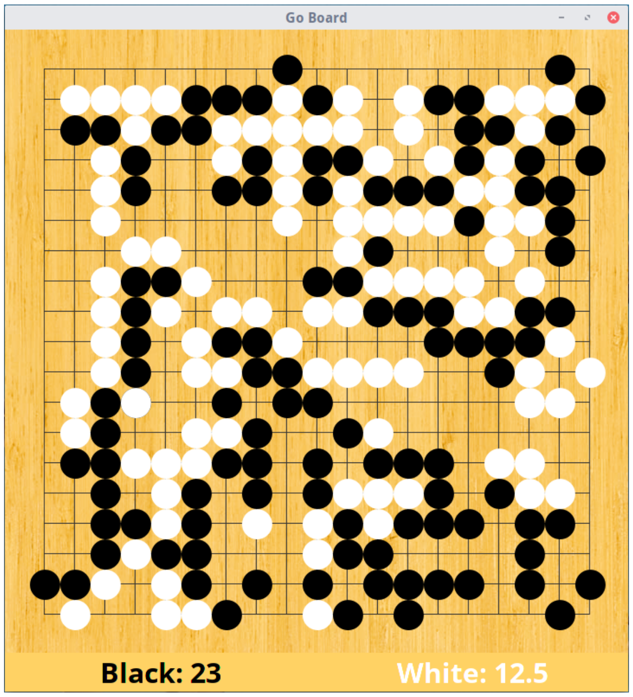
Next, draw an 18×18 grid of black squares by calling your DrawShape.drawRectangle() method in a for loop. Each square should have a width and height of 35, and the lower left corner of the entire grid should be at the position $(45, 45)$. **Hint:** Rather than creating 324 different Rectangle objects, you can use the move() method to slide it 35 pixels up and to the right between iterations.

Finally, iterate over every Color in the stones array. If the color at stones[i][j] is not null, then use your DrawShape.drawCircle() method to display it on the board. The circle should have a radius of 16, and its center at the position $(45 + i \cdot 35, 45 + j \cdot 35)$. Again, you can move() a single circle instead of creating hundreds of distinct objects.

[1] fromFile() is an example of a *factory method*. This is a common design patter in OOP – Using a static method to do some extra work that should be left out of the constructor.

Go Frame

`GoFrame.java` inherits from `JFrame`. The constructor takes a `GoBoard` as an argument, and adds it to the frame. In addition, it adds a `JPanel` which will display a final score for the game. The finished product should look like this:



Black: 23

White: 12.5

The visual properties to keep in mind:

- There are two `JLabels`, one on the left reads `"Black: 23"`, and another on the right says `"White: 12.5"`
- Both labels are center aligned (`SwingConstants.CENTER`), and have a bold, 32-point, sans-serif font
- The text of the left label is black, and the text of the right label is white
- The labels belong to a `JPanel` with a background color set to `"#ffd294"`
- The `JPanel` uses a grid layout, with 1 row and 2 columns
- The `JPanel` is underneath (`BorderLayout.SOUTH`) the central `GoBoard` component

Write a main() method that creates a new `GoBoard` from the file `stones.txt`. Create a new `GoFrame` from this `GoBoard`, and display the resulting window. The program should exit when the window is closed.

Submission

Make sure that you have Javadoc style comments for every class and method in your source code, as described in Homework 0. Document any unresolved bugs in the Javadoc comments for each of your classes. When you are satisfied that your code is complete, create a TAR file containing all of the source code for this assignment called:

```
cse213_<firstname>_<lastname>_hw7.tar.gz
```

Upload your submission to Canvas before the due date.